Homework 2: Emotion Classification with Neural Networks (100 points)

Kathleen McKeown, Fall 2019
COMS W4705: Natural Language Processing

Due 10/14/2019 at 11:59pm

Please post all clarification questions about this homework on the class Piazza under the "hw2" folder. You may post your question privately to the instructors if you wish. If your question includes code or a partial solution, please post it privately to the instructors ONLY.

This is an individual assignment. Although you may discuss the questions with other students, you should not discuss the answers with other students. All code and written answers must be entirely your own.

Late policy: You may use late days on this assignment. However, you MUST include at the top of your submission how many late days you are using. If you don't tell us or have used all your late days, 10% per late day will be deducted from the homework grade. You may choose to save your late days for harder assignments later in the class.

1 Introduction

This homework will serve as your introduction to using neural networks, a powerful tool in the world of NLP. You will explore neural networks through the problem of emotion classification, in which you assign one of several emotion labels to a piece of text (contrast this multi-class problem with your first homework, where you developed separate models for many binary problems). You will implement and apply different neural architectures to this problem and work through the math that defines them. The dataset you will use for this homework is taken from a CrowdFlower dataset[1] (Sentiment Analysis: Emotion in Text) from which a subset was used in an experiment published on Microsoft Azure AI Gallery[2].

[1] `https://www.figure-eight.com/data-for-everyone/`
[2] `https://gallery.azure.ai/Experiment/Logistic-Regression-for-Text-Classification-Sentiment-Analysis-1`

2 Homework Instructions

2.1 Programming Problems (48 points)

For this assignment, you will implement different popular neural network architectures and test them out on a social media emotion classification dataset. You will use a popular deep learning framework, PyTorch, to implement and train these neural models. For this homework, we will provide a GPU image on Google Cloud (`coms4705-gpu-student`; refer to the Google Cloud setup instructions if you need help finding it) within a few days of the homework's release. The networks in this assignment should run quickly even without GPUs, so if you prefer, you may install PyTorch on the CPU image (using `conda install pytorch cpuonly -c pytorch`; see the the download page) or work locally.

To get started, download the provided code from the website and open `hw2.py`. Ultimately, the provided code and the code you write should work together to load the text data from the `data/crowdflower_data.csv` file, preprocess it and place it into Pytorch `DataLoaders`, create a number of models, train them on the training and development data, and test them on a blind test set. Most of the code is already written for you; you will complete this assignment by filling in some code of your own.

Provided Data

The data consists of 27,471 Tweets which were labeled with emotion labels through crowdsourcing. We have selected the four emotion labels with the most data from the original dataset: 'neutral', 'happiness', 'worry', and 'sadness'. Each Tweet has been given exactly 1 of these labels, and they are not preprocessed in any way before you run `hw2.py`.

Provided Code

Code is already provided to 1) load, preprocess, and vectorize the data; 2) load pretrained 100-dimensional GloVe embeddings (trained on Twitter); and 3) test a generic model on the test set. The preprocessing code is located in `utils.py`. You may **not** modify the `test_model()` function at any time. You may modify other code (e.g., the preprocessing code) as part of your code extensions (see section 2.1.3).

The `main()` function in `hw2.py` is provided to start you out; it loads and preprocesses the data, and will save it to file if you set `FRESH_START = True` and load it if you set `FRESH_START = False`. You should use this function to run and test your code and to train the models you submit, although we will not grade you on it specifically.

2.1.1 Training Code

You will need to fill in the `train_model()` function in `hw2.py` to train your models. You may **not** modify the function header. The `train_model()` function you submit (and use to train your own models) should do the followimg:

- Take in an arbitrary created model, a created loss function, a created optimizer,

and the train and development data;

- Train by looping through minibatches of the whole training set (the provided `DataLoader`s already provide minibatches of size 128; you may change the batch size if you wish) (one pass through the whole training data is called an **epoch**);
- Zero the gradients before processing each minibatch;
- Calculate the predicted output on each minibatch;
- Calculate the loss on each minibatch (between the gold labels and your model output) using the existing loss function;
- Perform backpropogation with the loss from each minibatch;
- Take an optimizer step each time you perform backpropogation;
- At the end of each epoch, calculate the total loss on the development set and print it out;
- Train until the loss on the development set stops improving; and
- Return the trained model.

2.1.2 Base Models

You will implement two basic models, a dense neural network and a recurrent neural network, by filling in the `init()` and `forward()` functions for the `DenseNetwork` and `RecurrentNetwork` classes in the `models.py` file. You must submit these two trained models, saved using `torch.save()`; those saved models must conform to the architectures we lay out below and may not include any additional extensions.

When you save your models, make sure to

- Save the whole model, not just the state_dict, and
- Move the model to CPU before saving if you were working on a GPU.

Dense Network
Your dense network should...

- Feed the provided lists of word indices into an embedding layer initialized with the pretrained GloVe embeddings;
  - The embedding layer must be 100-dimensional to match the pretrained embeddings. You may choose whether to freeze the embedding layer or keep training it.
- Take the sum of all word embeddings for the sentence;
- Feed the result into a two-layer feedforward network, whose output is a vector of size 4;

  - You may add dropout and decide the hidden size in this netwoek.

Recurrent Network
Your recurrent network should...

- Perform the same embedding step as above (without summing; you should get a sequence of embeddings as output);
- Run the produced embeddings through a two-layer recurrent network (plain RNN, LSTM, or GRU); and
  - You may choose the type of RNN, the hidden size output by the RNN, the number of layers, and the dropout.
- Project the final hidden state of the RNN to a vector of size 4, as above (i.e., using a single dense layer).

2.1.3 Extensions

Finally, experiment with the architecture and training of your networks. Select **two** of the following and try them out on one or both of your networks:

- A different word embedding setting (using a different set of pretrained embeddings that were trained on an emotion-related task, training your own word embeddings on the corpus, etc.)
- Changes to the preprocessing of the data (tokenizers specifically for Tweets, a different method of selecting the vocabulary, etc.)
- Architecture changes (adding attention to your recurrent network, flattening embeddings using some method other than sum in the dense network, etc.)
- Or a similarly significant change that you come up with (i.e., no simple parameter tuning)

These extensions should all be present in the `hw2.py` and `models.py` you submit. You may try them out individually or both at once.

NOTE: You must submit your **ORIGINAL** code for the `DenseNetwork` and `RecurrentNetwork`; PyTorch requires us to have the original class definition to load your models. If you experiment with code inside one of the models, put that code in a **NEW** model class in `models.py`. If we cannot load your models, you will not receive points for their performance.

2.2 Written Problems (39 points)

Now that you have implemented these architectures in code, let's understand what's happening at a mathematical level. In this section, you will calculate the forward pass of a dense neural network and work through an example of backpropagation to understand how the network learns.

Submit the answers to these problems, with your work, in your typeset submission as described in Section 3.4. (You do not have to show work for multiplying two matrices together).

2.2.1 Dense Network - Forward (10 points total)

Suppose we have a simple dense network. The forward pass of this network is described by:

$$Z_1 = W_1 A_{in} + b_1 \tag{1}$$
$$A_1 = f(Z_1) \tag{2}$$
$$Z_{out} = W_{out} A_1 + b_{out} \tag{3}$$
$$A_{out} = f_{out}(Z_{out}) \tag{4}$$

where 1-2 describe a feed-forward layer and 3-4 describe the output layer.

Suppose that you are given the following:

$$W_1 = \begin{bmatrix} 1 & -1 & 2 & 3 & 0 \\ 4 & 0 & -1 & 1 & 3 \\ 2 & 1 & 3 & -5 & -4 \\ 4 & -3 & 2 & 1 & -3 \end{bmatrix} \quad b_1 = \begin{bmatrix} -1 \\ 2 \\ -4 \\ 3 \end{bmatrix}$$
$$W_{out} = \begin{bmatrix} 2 & -2 & -1 & 3 \\ -2 & 1 & -5 & 4 \end{bmatrix} \quad b_{out} = \begin{bmatrix} 12 \\ 3 \end{bmatrix}$$
$$A_{in} = \begin{bmatrix} 2 & 1 \\ 3 & 4 \\ 5 & 3 \\ 1 & 1 \\ 4 & 2 \end{bmatrix}$$
$$f_1(x) = f_{out}(x) = relu(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

That is, for simplicity, we will assume that f_1 and f_{out} are the same function.

1. Calculate the following: Z_1 (3 points), A_1 (2 points), Z_{out} (3 points), and A_{out} (2 points).

2.2.2 Backpropagation (15 points total)

When the network learns via stochastic gradient descent, each time we see a training example, we make our prediction, calculate the loss with respect to the gold label, and then improve the network's parameters. We calculate the proportion of the loss attributable to each parameter—the gradient of the loss with respect to that parameter—and move that parameter a tiny amount in the opposite direction.

This is described by:

Parameters: W, b
Inputs: $\hat{x}$
Iterate until convergence (for given learning rate η)
$W \leftarrow W - \eta \frac{\partial Loss}{\partial W} Loss(\hat{x})$
$b \leftarrow b - \eta \frac{\partial Loss}{\partial b} Loss(\hat{x})$

Notice that stochastic gradient descent updates all of the parameters of the model.

Consider a neural network (Network N) with 4 inputs (x_1, x_2, x_3, x_4) defined below. Note: this is not a dense network but it is still a neural network.

Network N
Inputs: x_1, x_2, x_3, x_4
Hidden units: $x_5 = f_5(x_1), \quad x_6 = f_6(x_2, x_3), \quad x_7 = f_7(x_4)$
Output unit: $x_8 = f_8(x_5, x_6, x_7)$
Given by:
$f_5(x_1) = \sigma(x_1) = \frac{1}{1+\exp(-x_1)}$
$f_6(x_2, x_3) = a * x_2 + b * x_3 + c * x_2 * x_3$
$f_7(x_4) = (x_4)^2 + d$
$f_8(x_5, x_6, x_7) = \frac{\exp(x_6)}{\sum_{i=5}^{7} \exp(x_i)}$
Where:
$a, b, c, d \in \mathbb{R}$ are learned parameters.
End Network

Note: $\exp(x) = e^x$. Suppose that

$$a = 3, \quad b = 4, \quad c = 2, \quad d = 2$$

$$x_1 = 0, \quad x_2 = 2, \quad x_3 = -1, \quad x_4 = 2$$

learning rate

$$\eta = 0.1$$

and

$$\frac{\partial Loss(x_1, x_2, x_3, x_4)}{\partial x_8} = 3.$$

Answer the following:

1. For $i = 1, \ldots, 7$, write the formula to calculate

$$\frac{\partial Loss}{\partial x_i}.$$

(1 point each)

2. Calculate

$$\frac{\partial Loss}{\partial a}, \quad \frac{\partial Loss}{\partial b}, \quad \frac{\partial Loss}{\partial c}, \quad \frac{\partial Loss}{\partial d}$$

and update the learned parameters a, b, c, d. (2 points for each partial derivative, 1 point for each update)

2.2.3 Coding Reflections (10 points)

Finally, answer the following questions about the programming portion of this assignment:

- What extensions did you try in section 2.1.3? Where in the code did you need to implement them, or what code implemented each? Why did you think each of them might improve your performance? What was the actual effect of each one, and why do you think that happened? (For each extension, write a short paragraph.) (10 points, 5 per extension)

3 Grading (WHAT YOU NEED TO SUBMIT!)

You will be graded based on the following:

3.1 Programming (48 points total)

Your deliverables are the following:

- Your completed `hw2.py` file (10 points total)
  - The `train_model()` function should have the same function header as in the provided code, should create an optimizer, should train on the training set until the development loss stops decreasing, and should print the development loss at the end of each epoch. This function may include any experiments from section 2.1.3 as long as it does the above. (10 points)
- Your completed `models.py` file (20 points total)
  - Your `DenseNetwork` class should be complete as per the specifications in 2.1.2. (10 points)
  - Your `RecurrentNetwork` class should be complete as per the specifications in 2.1.2.. (10 points)
- Your `DenseNetwork` and `RecurrentNetwork` models saved with `torch.save()` (**NOTE: You should save the whole model, not just the state dict; also, save a CPU version.**) (6 points total)
  - Your `DenseNetwork` model should accept the output of the provided DataLoaders as input, produce a (batch_size, 4) matrix of probabilities as output, and conform to the architecture specifications in section 2.1.2. (3 points)
  - Your `RecurrentNetwork` model should accept the output of the provided DataLoaders as input, produce a (batch_size, 4) matrix of probabilities as output, and conform to the architecture specifications in section 2.1.2. (3 points)
- Your extensions from section 2.1.3 should be present and correctly implemented (12 points total).

- The extensions may be anywhere in your code; you **MUST** tag them with a comment pointing them out.
- Each extension is worth 6 points. If you do more than two, you will be graded on the first two listed in your written answers. **Make sure you include the `utils.py` file!**

We will not grade you on your `main()` function or any other function unless you have changed it drastically, but you should use the existing setup in `main()` to run your experiments.

3.2 Software Engineering (includes documentation) (5 points)

Within Code Documentation

- Code should be documented in a meaningful way. This can mean expressive function/variable names as well as commenting.
- Informative method/function/variable names.
- Efficient implementation.

3.3 $\mathbf{F}_1$ Score (8 points total)

You will also be graded on the performance of both of your saved models using the provided `test_model()` function. You will receive **4 points** for each model if you achieve at least **40 points in macro $\mathbf{F}_1$ score** for that model. This threshold should not be difficult to achieve given the starting parameters, and should rather serve as a check that you implemented your model correctly and understand the code in order to tweak parameters.

3.4 Written Answers (39 points total)

Your deliverables are the following:

- A **hw2-written.pdf** file containing your name, email address, the homework number, and your answers for the written portion. If you are using any late days for this assignment, note them at the top of this file. The following is the point breakdown for each problem.
  1. Dense Network - Forward (10 points total)
  2. Backpropagation (19 points total)
  3. Coding Reflections (10 points total)

4 Submission Instructions

You should submit the following on CourseWorks:

- A zip file named <**YOUR-UNI**>**-hw2.zip**, for example, **ect2150_hw2.zip**. This should have exactly the files listed in the deliverables in section 3.1, all placed inside a `<YOUR_UNI>_hw2` directory. Include any external files needed to run your code (e.g., word embeddings we did not provide you). You do not need to include the provided data files or pretrained word embeddings in the zip.

You should submit the following on Gradescope:

- The **hw2-written.pdf** as described in section 3.4.

NOTES:

1. We **WILL NOT** grade code that does not run in **Python 3.6.9**, INCLUDING because of Python 2 print errors. We will grade your code **on the VM image we provided you**, so you should test it there before submitting on CourseWorks.
2. Handwritten solutions **WILL NOT** be graded. This includes pictures of handwritten answers inside the PDF. If you have concerns about typesetting, please talk to the TAs or post on Piazza.

5 Academic integrity

Copying or paraphrasing someone's work (code included), or permitting your own work to be copied or paraphrased, even if only in part, is not allowed, and will result in an automatic grade of 0 for the entire assignment or exam in which the copying or paraphrasing was done. Your grade should reflect your own work. If you believe you are going to have trouble completing an assignment, please talk to the instructor or a TA in advance of the due date.