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

Kathleen McKeown, Spring 2021
COMS W4705: Natural Language Processing

Due 2/17/2021 at 11:59pm ET

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

Overview

This homework will serve as your introduction to using neural networks, a powerful tool in NLP. You will be doing emotion classification, in which you assign one of several emotion labels to a piece of text. You will implement and apply different neural architectures to this problem using PyTorch and work through the math that defines them. The data you will use for this homework is taken from a CrowdFlower dataset[1] from which a subset was used in an experiment published on Microsoft Azure AI Gallery[2].

1 Programming (46 points)

To get started, download the provided code from the website. 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.

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

1.1 Provided Resources

The data in `data/crowdflower_data.csv` consists of tweets labeled with 4 emotion labels: 'neutral', 'happiness', 'worry', and 'sadness'. Each tweet has exactly one label. The data is not pre-processed in any way.

Code is already provided to 1) load, preprocess, and vectorize the data; 2) load pre-trained 100-dimensional GloVe embeddings ; and 3) test a generic model on the test set. The preprocessing code is located in `utils.py`. You may **not** modify `test_model()`.

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` so that you do not have to re-process the data every time you run the code. You should use this function to run and test your code.

1.2 Tasks

In this assignment you will need to do the following:

1. Fill in the `train_model()` function in `hw2.py` to train your models (§1.2.1)
2. Implement a feed-forward neural network (dense network)
3. Implement a recurrent neural network
4. Implement **2** extensions of your choice
5. Fill in `main()` in `hw2.py` to run your models

1.2.1 Training Code

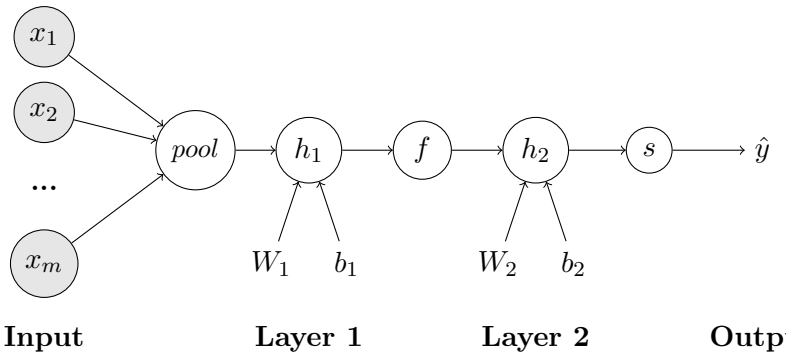
You will need to fill in the `train_model()` function in `hw2.py` to train your models. You may **not** modify the function header. The `train_model()` function you submit should do the following:

- Train by looping through minibatches of the whole training set;
- Calculate the loss on each minibatch (between the gold labels and your model output) using the existing loss function;
- Do backpropagation with the loss from each minibatch and take an optimizer step;
- At the end of each epoch, calculate and print the total loss on the development set;
- Train until the loss on the development set stops improving; and
- Return the trained model.

1.2.2 Models

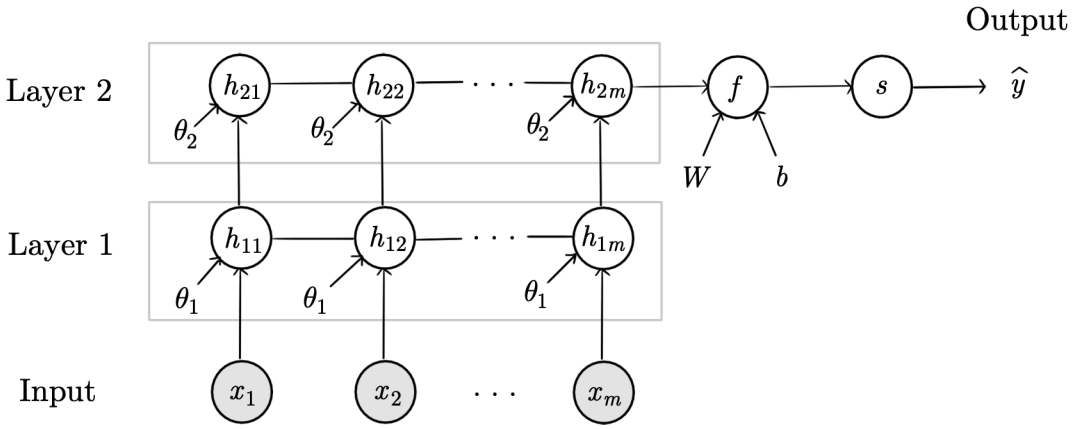
You will implement two basic models, a dense neural network and a recurrent neural network, by filling in the `init()` and `forward()` functions for the `DenseNetwork` and `RecurrentNetwork` classes in the `models.py` file.

Dense Network
Your dense network should follow the computation graph below:



where s is the $softmax$ function and *pool* is a function $g : \mathbb{R}^{d \times m} \rightarrow \mathbb{R}^d$ that takes as input a sequence of m vectors in $\mathbb{R}^d$ and outputs a single vector in $\mathbb{R}^d$. For example: max-pooling. Note that f is a non-linear function and $y \in \mathbb{R}^4$.

Recurrent Network
Your recurrent network should follow the computation graph below:



You can choose the type of RNN (plain RNN, LSTM, GRU).

1.2.3 Extensions

Finally, experiment with the architecture and training of your networks. Select **2** non-trivial extensions and create a **new network class** or classes to test them out.

NOTE: if you change the pre-processing, you **must** put this into different functions so that your original dense and recurrent networks run **without** the extensions.

Some examples of **non-trivial** (i.e., acceptable) extensions include:

- A different word embedding setting (using a different set of embeddings 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, etc.)
- Architecture changes (adding attention to your recurrent network, etc.)
- Different strategies for optimization and training (adding a learning rate scheduler, doing more complicated forms of early stopping, etc.)

Some examples of **trivial** (i.e., unacceptable) extensions include:

- Changing the non-linearity or pooling function
- Changing hidden sizes, batch size, etc.
- Adding dropout

These extensions should all be present in the `hw2.py` and `models.py` you submit.

2 Written Problems (54 points)

For the written part, you will do an exercise that will help you understand how neural networks work on a mathematical level. You will also do some homework problems related to part-of-speech tagging and syntax. There are four questions in this part:

1. Coding reflections (4 points total)
2. Backpropagation with RNNs (28 points total)
3. Context free grammar (12 points total)
4. Viterbi algorithm (10 points total)

Submit the answers to these problems, with your work, in your typeset submission as described in §3.2. (You do not have to show work for multiplying two matrices together).

2.1 Coding Reflections (4 points)

Answer the following questions about the programming portion of this assignment:

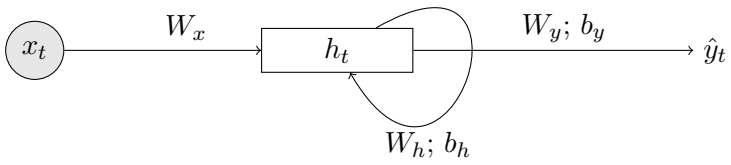
- What extensions did you try in §1.2.3? Where in the code did you need to implement them? Why did you think each of them might improve your performance? What was the actual effect of each one, and why do you think that happened? (For each extension, write a short paragraph.) (4 points, 2 per extension)

2.2 Backpropagation with RNNs (28 points total)

In this section, you will see how a recurrent neural network learns by working through one iteration of stochastic gradient descent (SGD).

2.2.1 Forward Propagation (11 points)

Suppose we have a simple recurrent neural network whose architecture is described as:



and which has the following parameter specifications:

$$\hat{y}_t = f(p_t) \qquad p_t = W_y^T h_t + b_y \qquad h_t = f(q_t) \qquad q_t = W_x^T x_t + W_h^T h_{t-1} + b_h$$

where $x_t \in \mathbb{R}$, $h_t \in \mathbb{R}^2$, $\hat{y}_t \in \mathbb{R}$, f is the sigmoid activation function specified as $f(x) = \frac{1}{1+e^{-x}}$, and

$$W_x = \begin{bmatrix} 1 & 3 \end{bmatrix} \qquad W_y = \begin{bmatrix} 3 \\ 1 \end{bmatrix} \qquad W_h = \begin{bmatrix} 1 & 3 \\ 2 & 1 \end{bmatrix} \qquad b_h = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \qquad b_y = \begin{bmatrix} 1 \end{bmatrix} \qquad h_0 = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

Questions:
The first step in SGD is to predict the RNN's output on a given input via forward propagation. Suppose we have an input $X = \begin{bmatrix} -1 & 0 \end{bmatrix}$, such that $x_{t=1} = -1$ and $x_{t=2} = 0$. Let the corresponding gold labels be $Y = \begin{bmatrix} 0 & 1 \end{bmatrix}$, that is $y_{t=1} = 0$ and $y_{t=2} = 1$.

1. **(10 points)** Calculate the output and hidden state of the RNN (Hint: Consider how the network unfolds over time). Specifically, calculate the following:
   a) $h_{t=1}$ (2 points)
   b) $\hat{y}_{t=1}$ (2 points)
   c) $h_{t=2}$ (3 points)
   d) $\hat{y}_{t=2}$ (3 points)

2. **(1 point)** Now that we have the predicted output of the RNN, we determine the 'error' or loss with respect to the gold labels. We will use the Mean Squared Error (MSE) loss function:

$$\mathcal{L}_{MSE} = \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{m_i}(y_t^{(i)} - \hat{y_t}^{(i)})^2$$

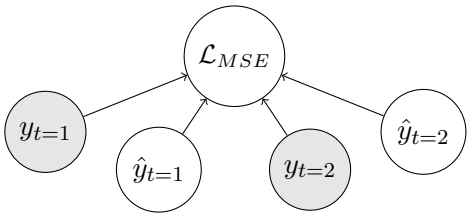
where N is the number of examples, m_i is the sequence length for example i, $\hat{y_t}^{(i)}$ is the predicted label for example i at timestep t, and $y_t^{(i)}$ is the gold label for example i at timestep t.

2.2.2 Backpropagation Through Time (17 points)

Backpropagation is the key to learning via SGD. For recurrent networks, we use a specific form of backpropagation, called backpropagation through time (BPTT), to compute gradients of network parameters with respect to the loss. In order to do BPTT, we must first expand the computational graph of the RNN in order to obtain the dependencies among model parameters. We say a parameter a is dependent on another parameter b if b is used to calculate a during forward propagation. Once we determine the dependencies between parameters and calculate gradients with respect to the loss, the final step during learning is to update the network's parameters.

Questions:

1. **(4 points)** Draw the computation graph of this RNN.
   Hint: We have provided the first step of the graph; fill in the rest.



2. **(10 points; 2 points each)** Calculate the gradients with respect to the parameters of the RNN. You **must show your work** and your answers should be numerical (not equations). Specifically, calculate the following:

$$\text{a) } \frac{\partial \mathcal{L}_{MSE}}{\partial W_y}; \quad \text{b) } \frac{\partial \mathcal{L}_{MSE}}{\partial W_h}; \quad \text{c) } \frac{\partial \mathcal{L}_{MSE}}{\partial W_x}; \quad \text{d) } \frac{\partial \mathcal{L}_{MSE}}{\partial b_y}; \quad \text{e) } \frac{\partial \mathcal{L}_{MSE}}{\partial b_h}$$

3. **(3 points)** Update the following parameters of our RNN. Recall that the standard SGD update is

$$W \leftarrow W - \eta \frac{\partial \mathcal{L}_{MSE}}{\partial W}$$

Let the learning rate be $\eta = 0.01$. Specifically, update

a) b_y (1 point)

b) W_h (2 points)

2.3 Grammar (12 points)

(a) The soft white snow fell quietly in the city on Monday.

(b) The snow in the park drifted to 3 feet.

(c) The small boy built a very large round snowman who wore a red cap.

(d) The boy gave the snowman a carrot nose.

1. **(8 points)** Provide a context-free grammar that can be used to parse sentences (a) through (d). Your grammar should provide at least one parse tree that captures that correct meaning of each sentence. Ideally, your rules would handle similar sentences not in the list below.

   Choose POS tags for your rules from the Universal Dependencies tagset (Fig. 8.1, Chapter 8, p 2, Jurafsky and Martin).

   Keep in mind that siblings on the right hand side of a rule indicate modification and the right hand side provides the sub-structure of the non-terminal on the left hand side. You can use recursion to allow for multiple modifiers. Modification should be represented correctly in the generated parse tree.

   NOTE: Your grammar should consist of 18 rules or less. It should not allow ungrammatical sentences to be generated.

2. **(4 points)** Show all possible parse trees that allow your grammar to generate sentence (a).

2.4 Viterbi Algorithm (10 points)

	I	want	fly	the	off	sandwich	my
VB	0	.25	.45	0	0	0	0
TO	0	0	0	.01	0	0	.01
NN	0	.2	.2	0	0	.4	.01
PPSS	.40	0	0	0	0	0	.01
DT	0	0	0	.5	0	0	0
PRP$	0	0	0	0	0	0	.5
IN	0	0	0	0	.8	0	0

Table 1: Observation Likelihoods

Given the sentence "I want the fly off my sandwich." show how you would compute the probability of "fly" as a verb versus the probability of "fly" as a noun in the context of this sentence using the probabilities in Tables 1 and 2 and the Viterbi algorithm. For this question you should:

	VB	TO	NN	PPSS	DT	PRP$	IN
<s>	.018	.0042	.041	.067	.068	.068	.03
VB	.0038	.035	.047	.0070	.0070	.0070	.003
TO	.83	0	.0047	0	.0042	.0042	0
NN	.0040	.016	.087	.0045	.001	.001	.004
PPSS	.83	.0079	.0012	.0014	.001	.001	.001
DT	0	0	.85	0	0	0	0
PRP$	0	0	.85	0	0	0	0
IN	.003	.001	.04	.003	.05	.05	0

Table 2: Tag transition probabilities. The rows are labeled with the conditioning event. Thus, P(VB|<s>) = .018.

1. **(8 points)** Show the dynamic programming trellis at each state up to the point where "fly" is disambiguated.
2. **(2 points)** Show the formula with the values that would be used to compute the probability of "fly" as either verb or noun. You do not need to do the arithmetic. Just show the formula that would be computed.

3 Deliverables and Submission Instructions

PLEASE READ:

- **We WILL NOT accept code that does not run in Python 3.6.***; this includes broken code because of Python 2 print errors.
- **We WILL NOT accept hand-written work** for the written portion **EXCEPT** for the computation graph, trellis, and parse trees.
- If we cannot find your extensions with `CTRL+F` on `extension-grading` you will lose points.

3.1 Programming (46 points total)

Submit **one** zip file named **<YOUR-UNI>_hw2.zip** to CourseWorks.
This should have **exactly** the following files. Please DO NOT include the data files in the zip. Include any external files needed to run your code (e.g., word embeddings we did not provide you).

Your zip file should have at least the following:

- (8 points total) `hw2.py` including:
  - (8 points) The `train_model()` function as in §1.2.1.

  - A `main()` function that runs your dense and recurrent models **without any extensions**, and then runs your extensions separately. This will be used to grade your F_1 score.

- (18 points total) `models.py`, including:
  - (6 points) Your `DenseNetwork` as in section §1.2.2.
  - (12 points) Your `RecurrentNetwork` class as in §1.2.2.
- (14 points total) **Extensions**:
  - The extensions may be anywhere in your code; you **MUST** tag them with a comment pointing them out, and include the verbatim text "`extension-grading`" in that comment, or they **will not be graded**!
  - Each extension is worth 7 points. If you do more than two, you will be graded on the first two listed in your written answers.
- `utils.py`: As provided, you do not need to change this.
- (2 points total) **Documentation:** Your code should be documented in a meaningful way and implemented efficiently. This can mean expressive function/variable names as well as informative comments.
- (4 points) $\boldsymbol{F_1}$ **Score**: You will also be graded on the performance of your dense and recurrent models using the provided `test_model()` function. You will receive **2 points** for each model if you achieve at least **40 points in macro $\mathbf{F_1}$ score** for that model.

3.2 Written Answers (54 points total)

You should submit the following on Gradescope:

- 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. This file should include:
  1. Coding Reflections (4 points)
  2. Recurrent Neural Network (28 points)
  3. Context-Free Grammar (12 points)
  4. Viterbi Algorithm (10 points)

4 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. Note that for the programming portion:

- You **may** consult internet tutorials and official package source code (e.g., PyTorch) but you may not copy ANY PORTION without citation.
- You **may NOT** consult any other code including: Github repositories and other students' solutions.
- You **may NOT** post your homework solutions publicly on Github or allow another student to see any portion of your solution.