Assignment 2

The assignment consists of 3 parts, each giving up to 5 points. So, the maximum points for this assignment is 15.

I remind you that the assignment must be done alone. Of course, you may consult with each other but the actual work must be done alone.

Submit your answers to the question via Moodle in plain text.

Part 1

We have an input vector x, containing 10 elements [1.0, 2.0, ...,10.0]. A function f(x) returns the maximum value of the input vector. Our target value for f(x) is 12.0. We use absolute error as the loss function to describe how far we are from our target with our current input x. Use pytorch to find the gradients for the elements of x, using the given target value and loss function. Copy paste the code and also give the answer in plain words.

Torch API docs: http://pytorch.org/docs/master/torch.html

Points: 5

Part 2

Using the lab notebook from April 6 (http://nbviewer.jupyter.org/urls/dl.dropbox.com/s/raeg8n1lxobhzfr/Lab09.ipynb) as a reference, implement two text classification models, CbowSum and CbowMean (CBOW referes to continious bag-of-words).

The CbowSum model works like that: first, word embeddings are looked up for all words in the document. Then, the word embeddings are summed over the words of each document. The sum is passed through a hidden layer with 100 units (using the ReLU non-linearuty), then a dropout layer and the result goes to the final linear classification layer. Hint: use the `torch.sum()` function with appropriate arguments to sum the embeddings.

The CbowMean model is just like CbowSum, but the word embeddings are averaged over the words in the document, not summed as in CbowSum.

The models should have the same structure as the CnnText model, i.e., it should be possible to use the same training function to train and evaluate the models:

```
class CbowSum(nn.Module):

  def __init__(self, num_classes, vocab_size, embedding_dim, dropout_prob):
    super(CbowSum, self).__init__()
    # Implement this
```

```
  def forward(self, x):
    # Implement this
    return logit
```

Submission should include the source code of the models (i.e., the above implmentations for the two models), and nothing else.

Points: 5

Part 3

You have now trained several models for sentiment analysis on the same data, each with a slightly different architecture. Often it's beneficial to interpolate the predictions of several models for best performance. Interpolation (in our context) is just a fancy name for taking the average over the model predictions. E.g. if model A assigns probabilities [0.1, 0.9] to an input, and model B assigns [0.5, 0.5], then their interpolation (with equal weights) is `([0.1, 0.9] + [0.5, 0.5]) / 2.0 = [0.3, 0.7]`.

Your task is to implement a function `evaluate_interpolated(data_iter, models)`. The models parameter is a list of trained models (e.g. `[model1, model2]`). The function should return the accuracy of the interpolated predictions, using uniform weights. Note that interpolation should be applied to normalized probabilities, not to raw outputs that the model returns. Hint: use `F.softmax(model(text), dim=1)` to normalize the raw outputs.

Template for the function is here:

```
def evaluate_interpolated(data_iter, models):

  # Activate the evaluation mode for models. For example, this turns off dropout
  for model in models:
    model.eval()

  # Implement interpolation of model prediction and compute accuracy

  # return the interpolated average accuracy
  return accuracy
```

Test the function by giving it the original CnnText model and the CbowSum and CbowMean models you implemented in this assignment. What is the resulting accuracy?

Submit: copy/paste your commented implementation of `evaluate_interpolated()`, and the resulting accuracy of 3-model interpolation.

Points: 5